03｜我们要怎么理解领域驱动设计？

2021-06-29 徐昊

《如何落地业务建模》 课程介绍 >

讲述：徐昊

时长 20:35 大小 18.85M

你好，我是徐昊。今天我们来聊聊领域驱动设计中的提练知识的循环。

在上一讲，我们介绍了什么是统一语言，讲了它为什么是领域驱动设计的必要实践，以及为什么统一语言提供了一种更好的协同方式的可能性。

在回答这些问题的过程中，我们强调了统一语言和它背后的领域模型，赋予了研发人员通过重构定义业务的能力。这是从技术方的角度来理解的统一语言。那么**从业务方的角度来看，要如何利用统一语言去影响研发方呢**？

那就要谈到“两关联一循环”里的“一循环”了：提炼知识的循环。它是**领域驱动设计的核心流程**。

将提炼知识的循环看作开发流程

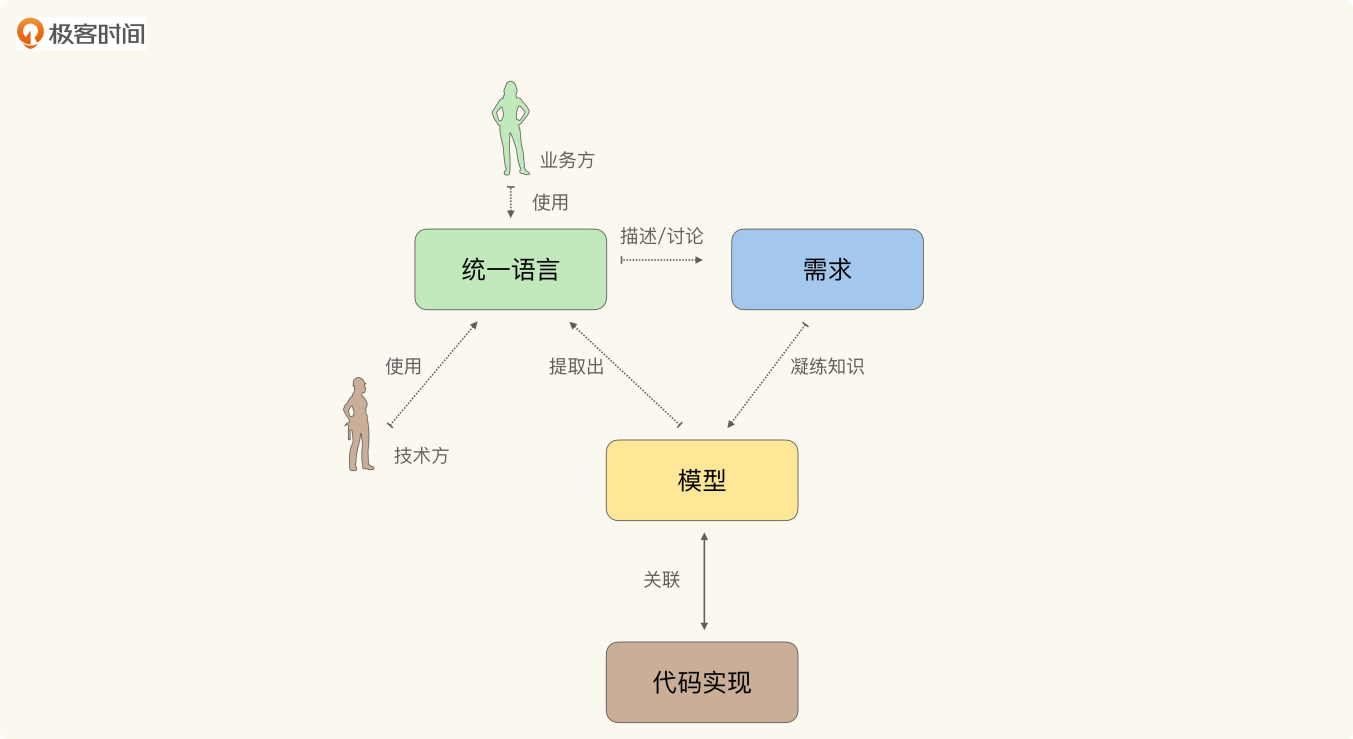
上节课已经说过，提炼知识的循环大致是这样一个流程：

首先，通过统一语言讨论需求；

而后，发现模型中的缺失或者不恰当的概念，然后精炼模型，以反映业务的实际情况；

接着，对模型的修改会引发统一语言的改变，再以试验和头脑风暴的态度，使用新的语言以验证模型的准确。

如此循环往复，不断完善模型与统一语言。示意图如下：



如果我们仔细思考一下上面的场景，会发现当模型中出现不恰当的概念时，提炼知识的循环就和**重构**（Refactoring）的过程有诸多相似之处：

1. 发现坏味道（Bad Smells）以明确改进方向：头脑风暴与试验，通过统一语言描述需求，发现模型中存在不恰当不准确的概念；
2. 尝试消除坏味道以改进目前状况：修改模型，提炼知识到模型中；
3. 通过坏味道是否消失判断改进是否成功：提取统一语言，头脑风暴与试验，验证新的模型是否准确。

而发现存在缺失概念的过程，也与**测试驱动开发**（Test-Driven Development）的过程相去不远：

1. 构造一个失败测试表明需求的变化：头脑风暴与试验，是否存在统一语言无法描述的业务；
2. 修改代码实现：修改模型，提炼知识到模型中；
3. 通过测试以证明需求变化完成：提取统一语言，头脑风暴与试验，验证新的模型是否准确。

你看，无论我们将提炼知识的循环看作通过统一语言对模型的重构，还是通过统一语言对模型的测试驱动开发，我们都可以把它看作**以模型为最终产出物的研发流程**。如果是重构，业务方就是那个发现坏味道的人；而如果是测试驱动开发，业务方就是构造测试的那个人。

让我们看一个例子，以便对这个流程有个更直接的认识。比如在上一讲极客时间的例子里，我们的统一语言是：

> 用户（User）是指所有在极客时间的注册过的人；
>
> 订阅的专栏（Subscription）是指用户付费过的专栏；
>
> 用户可以订阅多个专栏；
>
> 订阅。

并使用这个语言描述了需求：

> 作为一个用户（User），当我查阅购买过的专栏（Subscription）时，从而可以看到其中的教学内容。

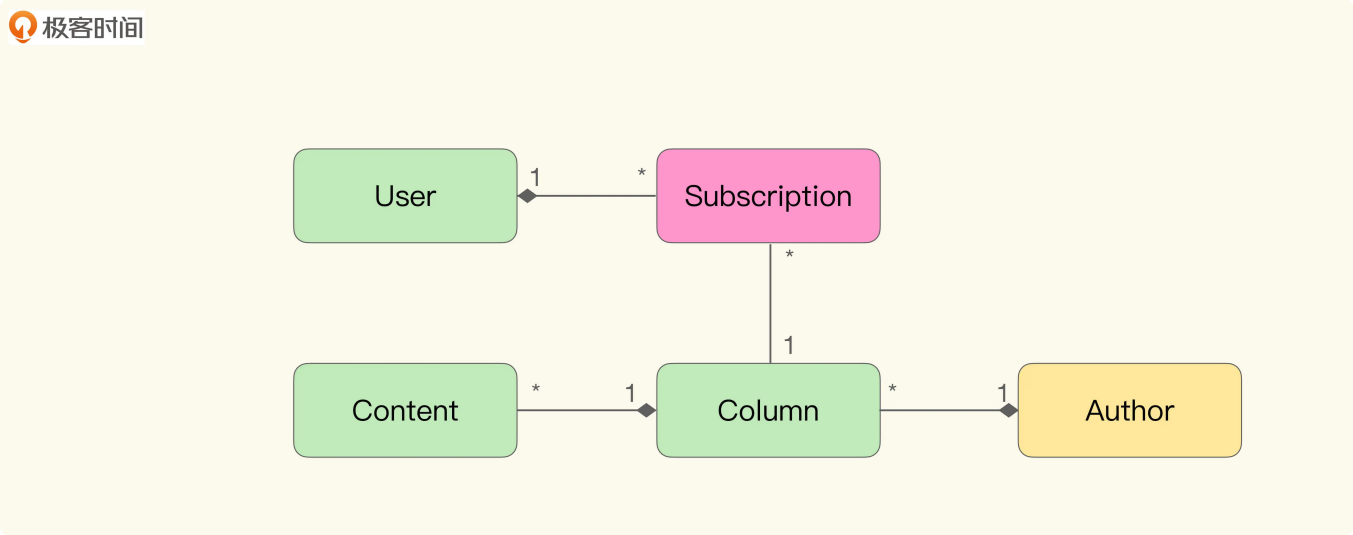
> 当用户（User）已购买过了某个专栏（Subscription），那么当他 / 她访问这个专栏时，就不需要再为内容付费。

我们可以听到用户反复提及**专栏**与**内容**。而在我们的语言中，只存在**购买过的专栏**（Subscription）。如果业务方想要描述用户没有购买的专栏，该怎么办？比如有这样一个需求：

> 作为一个用户（User），当我对某个专栏的内容感兴趣时，我可以购买这个专栏，使其成为我购买过的专栏（Subscription）。

于是在这条业务描述中，就出现了我们无法表达的含义：专栏和内容。这表示着我们的统一语言存在无法表达的业务，我们的模型存在漏洞。

当然，想弥补漏洞也是很简单的。我们可以联合团队一起**进行头脑风暴，提取缺失的概念，修改已有的模型**，并**由业务方判断这个修改后的模型是不是足够准确地表达了业务概念与逻辑**。那么最后，我们可能会得到这样一个新的模型：



然后我们可以按照第二讲讲过的内容，从领域对象与逻辑、界限上下文、系统隐喻等角度，重新提取统一语言。这里，我们假设在模型上又提取了**课程发布的界限上下文**。于是我们的统一语言就变为一个更丰富的形式：

> 用户（User）是指所有在极客时间注册过的人；
>
> 专栏（Column）是一组付费内容（Content）的集合，由极客时间签约的作者（Author）提供；
>
> 付费内容（Content）是课程的承载，可以是文字、视频或者音频；
>
> 作者（Author）是极客时间寻找的在某些领域有经验与专长的实践者；
>
> 订阅的专栏（Subscription）是指用户付费过的专栏；
>
> 用户可以订阅多个专栏；
>
> 专栏中可以包含多个付费内容；

同一作者可以发布多个专栏；

订阅；

课程发布。

于是之前无法被描绘的业务逻辑变成了这样：

> 作为一个用户（User），当我对某个专栏（Column）的内容（Content）感兴趣时，我可以购买这个专栏，使其成为我购买过的专栏（Subscription）。

在这样的业务描述中，就没有统一语言之外的概念了。至此，我们就通过修改领域模型，改变了统一语言，结束了提炼知识的循环。然后我们就可以根据修改后的模型，去改变对应的软件实现。这样，就可以保证我们的模型与软件实现在提炼知识的循环之后，仍然是关联的。

研发方与业务方的协同效应

如果我们将提炼知识的循环看作以模型为最终产出物的研发流程，那么此时可以再考虑一下模型与软件实现的关联：**对代码的修改即是对模型的修改；对模型的修改也就是对代码的修改**。

要知道，提炼知识的循环最终会反映到具体的软件实现上，那么业务方实际也就更深入地参与到了研发流程中。如果说统一语言与模型的关联赋予了技术方定义业务的权利，那么**提炼知识的循环也就赋予了业务方影响软件实现的权利**。

讲到这儿，我们就可以综合统一语言，从一个更全面的角度来看待双方的**权利与义务**。

对于技术方来说，通过统一语言获得了定义业务的权利，但是同时也必须承担在提炼知识的循环中接受业务影响实现的义务；

对于业务方来说，提炼知识的循环赋予了他们影响软件实现的权利，同样也有接受研发方通过统一语言定义业务概念的义务。

就这样，知识消化以一种**权责明确**的方式，让业务方与技术方参与到对方的工作中。同时也在整体上，**给予了业务方和技术方一种更好的协同方式**。

你可能会问，为什么让业务方与技术方参与到对方的工作中，就是一种更好的方式呢？首先，软件开发的核心难度就在于**处理隐藏在业务知识中的复杂度**。

想要处理这种复杂度，首先需要**打破知识壁垒**（统一语言），如果双方对彼此的知识域有基础的了解（模型），那么知识传递与沟通的效率就可以变得更高。

其次，对于复杂的问题，需要**快速的反馈周期（提炼知识的循环）试错**。复杂的问题也就是没有现成答案的问题，只能快速试错。双方参与到彼此工作中，在流程中就可以给予对方反馈，而不是等到产生最终结果之后再评价。那么反馈速度自然就更快捷，同时浪费（研发时间、无效代码）也更少。

因而，**让业务方与技术方参与到对方的工作中，就在双方之间带来了更好的协同效应**（Synergy Effect），也就是 1+1>2 的效果，这正是知识消化最吸引人的地方。

那么，在我们完全了解了知识消化的“**两关联一循环**”之后，你会发现它们是**一个有机的整体**：

> 统一语言与提炼知识循环重新定义了业务方与技术方的权责，并倡导了一种更好的协同方式；
>
> 模型与软件实现关联是保证这种协同方法的前提条件和实现保证。
>
> 三方中缺少任何一个，“知识消化”都会失败。

所以我们要怎么理解领域驱动设计？

至此，我相信你对领域驱动设计以及“知识消化”法就有了相当的认识和了解。然而只要你搜一搜网上的讨论，很快又会陷入到深深的疑问中：我到底懂没懂？为什么别人说的跟我想的不太一样呢？

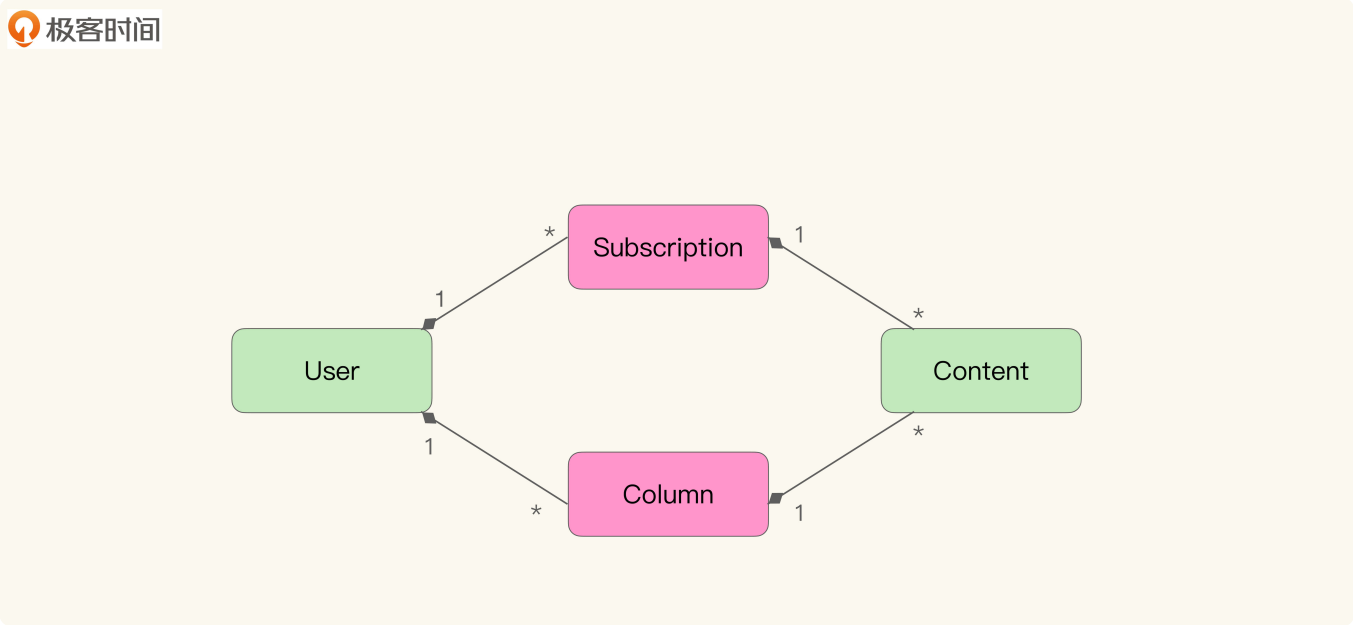
我们知道，“领域驱动设计”至少可以指代一种建模法，一种协同工作方式和一种价值观，以及上述三种按照随意比例的混合。因此，为了不让自己陷入诸多疑问的泥沼，我们还需要明辨“**当我们谈论领域驱动设计的时候，我们到底在说什么**”。

迭代式试错建模法

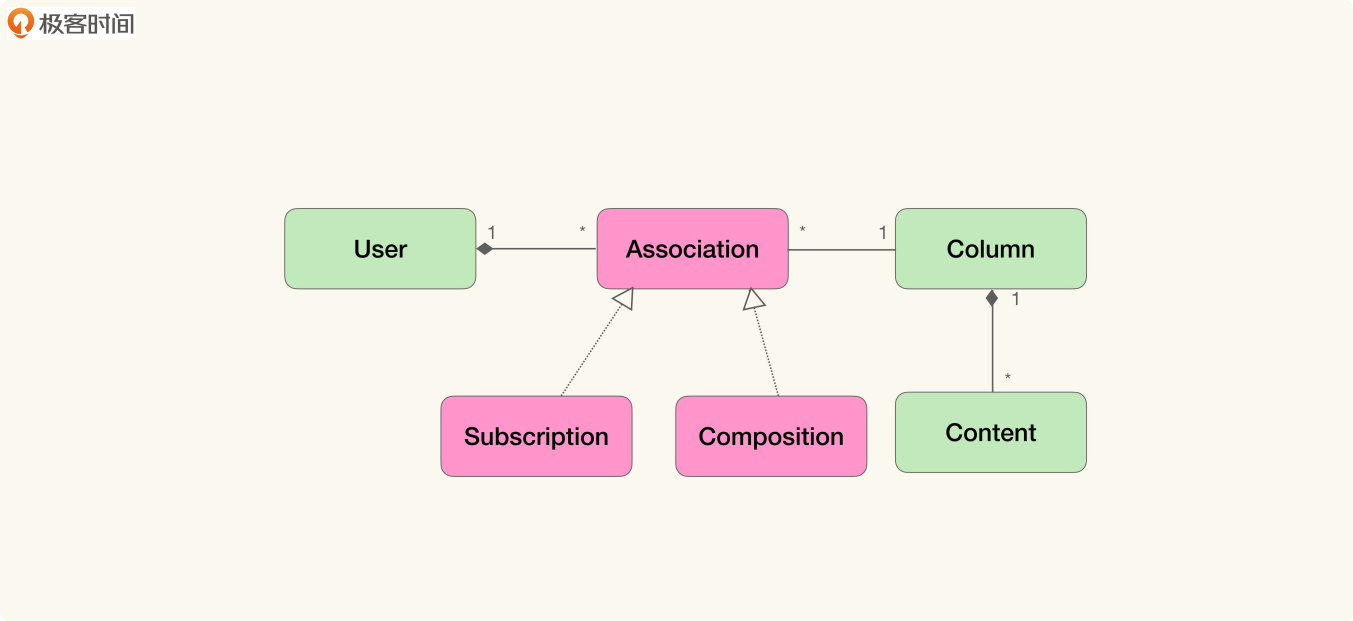
在理解了“两关联一循环”之后，也就理解了我们最开始提及的知识消化是一种**迭代式试错法**。然后你就会惊奇地发现，领域驱动设计作为一种名字里有设计方法，本身却很少谈及具体的模型设计，反而更偏向依靠交互来协同试错。

那么我们不禁要问，怎么说来说去都没有提到“**怎么才能保证模型的成功**”这个问题呢？

比如刚才讲解的极客时间专栏的例子，同样是发现专栏（Column）与内容（Content）的缺失，那么我们可以将订阅（Subscription）看作专栏的一个特殊状态。于是，似乎如此来建模也可以接受：



我们始终要记住，**模型是对问题的抽象，没有对错，只有角度不同**。上图里，我们把它当作一个对象的不同的状态。而在之前的例子里，我们把 Subscription 当作一种关联对象（Association Object），表示某个动作，也就是“订阅 Subscribe”的结果。我们还可以沿着这个思路建模：



那到底哪个模型更好呢？不好说，要看在具体需求上哪一个能更好地应对变化。

当然这也是所有试错法都存在的问题。正如 Eric Evans 自己所说：“**知识消化是一个探索的过程，你不可能知道你将会在哪里停止。**”其实后面还有他没直接说的一句话：“**你可能还不知道当你停止时，得到的是垃圾还是宝藏。**”这部分只能交给建模者的抽象能力，然后希冀一个好的结果。

总结来说，领域驱动设计之所以变成了如此不靠谱的建模法，是因为它尝试解决的问题域是复杂问题，也就是没有现成答案的问题。那么**迭代式试错法就是唯一“可行（Viable）”的方式。**

具有协同效应的工作方式

在这节课，我们花了大量的篇幅来解释为什么统一语言与提取知识的循环可以构成一种具有协同效应的工作方式。

简单总结一下，首先是**权利与义务的对等构成了协同的基础。**技术方通过统一语言获得了定义业务的权利，但是同时也必须承担在提炼知识的循环中接受业务影响实现的义务；而业务方呢，通过提炼知识的循环获得了影响软件实现的权利，但是同样，也有接受技术方通过统一语言定义的业务概念的义务。

在这种基础之上呢，权责明确的方式就形成了一种没有绝对主导的合作关系，让业务方与技术方都能参与到对方的工作中，从而产生 1+1>2 的协同效应。

不过这种协同方式是否能够发挥作用，很大程度上依赖于团队对模型，以及与其对应的统一语言的理解与接纳。这部分就依赖于建模者的变革管理能力。正如我们前面看到的，建模者可以很容易地从模型中提取统一语言提案，只要给所有概念明确的定义和边界就可以了。

而能否真正地变成统一语言，则需要团队接纳并逐步在工作中使用它。这是**行为改变，需要变革管理去推动**。而知识消化希望通过**头脑风暴与试验**的方法，简化这种变革。不过实际结果，仍然是因人而异的，没办法保证百分之百的成功。

价值观体系

如果我们看一下**领域驱动设计实践背后的价值观**，那么会发现：

领域驱动设计是一种模型驱动的设计方法：模型应处在核心；

两关联一循环：业务与技术围绕着模型的协同。

只要接纳这两条价值观的都可以声称自己采用了“领域驱动设计”。从这个角度来说，“领域驱动设计”是一个如此宽泛的定义，以至于我们可以使用“领域驱动设计”去讨论任何事情。

这是一个完美的例子，告诉我们要慎重对待抽象。毕竟到了一定的抽象程度，差异就被完全隐藏了。因而我会更建议你**使用“知识消化”来代替泛化的“领域驱动设计”**，毕竟在知识消化中，除了“模型应处在核心”以及“业务与技术围绕着模型的协同”外，它还框定了具体的实践架构：两关联一循环。

所以当我谈论“知识消化”的时候，我们就可以更好地去探讨这些问题：怎么更好地构建“富含知识的模型”？如何持续保持模型与软件实现关联？怎么有效地提取领域语言？如何推动业务方更主动地参与提炼知识的循环？

因而在这个章节后续的课程中，当我谈到了“领域驱动设计”，如果没有特别强调，那么我所指的就是“知识消化”。

小结

“领域驱动设计”至少可以指代一种建模法，一种协同工作方式和一种价值观。作为建模法，领域驱动设计是迭代改进试错法。这是一种**保底可行，但可能效率不高**的建模方法。如果对所处理的领域没有更高的认知，那么使用它起码不会错，只要给予足够的时间，总会成功的。

作为一种协同工作方式，领域驱动设计**提供的思路是相当精彩的**，所以也对行业产生了深远的影响，特别是统一语言。

而作为价值观来讲，则过于宽泛了，领域驱动设计之滥觞大抵源于此吧。

对于领域驱动设计和知识消化，多年来行业也多有诟病，主要集中在这么三点：

第一，随着 ORM 的流行，关联模型与实现慢慢成为行业内的默认做法，于是大家的关注点也逐步过渡到怎样才能更好地建模。但是**迭代试错效率真的不高**，特别是技术方与业务方没有足够的信任的时候，可能就迭代不起来了。

第二，统一语言虽然赋能了技术侧，使其有了话语权，但是也**给了技术人员不去学习业务的借口**。比如技术可以硬拗很多奇怪的东西给业务方，以此来逃避对业务的真正学习。

第三，统一语言并不容易实现，技术人员不习惯去驱动这样的改变。而**一旦无法形成真的统一语言，提炼知识的循环也就无法进行了**，那么整个方法也就失败了。

除此之外，我们也可以发现，知识消化法其实**更适合敏捷团队**。无论是通过统一语言协同交互，还是提炼知识的循环，都**需要对这种跨工种协同以及渐进式改进方式有足够的认同**。在敏捷方法实施较好的团队，这些都不是问题。而在其他情况下，则会有较大的变革成本。

目前，领域驱动设计和知识消化法更多地被当作**一种框架性方法**来使用，而其中的具体的实践，在行业中也多有改进。这些我们后续将会讲到。

思考题

通过前三讲的学习，你可能已经发现了，知识消化是一个逻辑性很强的领域驱动设计方法，任何一个环节出现纰漏都可能造成整个过程的失败。那么在关联模型与软件实现这个环节上，你能想象会出现什么困难吗？如果是你，会怎么处理？

欢迎把你的思考和想法分享在留言区，我会和你交流。同时，我也会把其中不错的回答在留言区置顶，供大家学习讨论。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 02 | 统一语言是必要的吗？

下一篇 04 | 跨越现实的障碍（上）：要性能还是要模型？

精选留言 (9)

写留言

Jxin 置顶

2021-06-30

关于内容：
只谈责权行动是难以维持的。责权利心法，责任权利利益必须匹配。
关不关心生产实现，对于业务方是一场无悬念的博弈。

关心了，成效很模糊，但累和困难很明确。因为我除了自己的业务还要学习开发定义的业务概念（学习/沟通成本），同时描述业务故事也得基于这些概念来沟通（约束），搞错...
展开

作者回复: 驱动变革的底线在于“这种做法有道理”。而难道在于制造紧迫感，紧迫感的制造是一种艺术

2 3

赵晏龙 置顶
2021-07-01

上一课我的回答基本都在这一课中得到了验证，看来我的理解和主流观点应该没有太大偏差，老师总结得还是更到位！

在大约9年多的学习和实践中（旧约实践，非常期待你的新约课程），我遇到的最大的问题应该是模型变更前后数据迁移成本较高，必须是人工迁移，并且生产环境无缝迁移基本...
展开

作者回复: 可以练 慢慢就多了

2

Oops! 置顶
2021-06-29

看到关于权责的论述的部分时脑海里浮现出一个疑问，就和让业务和技术双方都接受这个权责分配？很多时候，业务方只关心结果，其实没有意愿“享受”“影响软件实现”的权利，因此，就没有意愿承担相应的义务。本课后半程也提出了这样的问题，“如何推动业务方更主动地参与提炼知识的循环？”。万事开头难，这个可能是架构师在推动DDD落地过程中首先可能所遇到的坎——如何有效的启动知识消化的循环？这块可能和DDD的核...
展开

作者回复: 7-9节 通过不同的建模方法 增加参与感

1

webmin
2021-07-02

某知识服务商有一道考验新人试题，请说说怎么解读《新华字典》。

如果采用总结提炼划重点方法，你说它的重点在哪儿？哪个字重要，哪个字不重要？它的结构是完全平铺的，没有重点。

...

展开

作者回复: nice

4

箭和方糖

2021-06-29

回答思考题，模型本身是与技术无关的，但软件实现是受限于技术选型的，因此有可能出现模型不能很好的用所选技术直接实现，或技术本身的最佳实践会给模型引入一些新的概念，进而丰富统一语言，如观察者，访问者等等可能业务方需要适应的概念。

解决方法，我觉得还是权衡吧，如果技术对模型的反馈是大家都能够理解并接受的，未...

展开

作者回复: cache layer不一定全能解决

1

garlic

2021-06-29

可能的困难：业务能力差，技术人员不懂业务。业务方能力差，可以请一些业务专家来支持；技术人员不了解业务，组织一些专题培训，多向业务人员请教，不一定所有技术人员都要很懂业务，但是核心技术人员一定要懂，否则大概率会出问题。我是做外包项目的，基本是二次开发，业务功能通过现有的系统或多个系统组合实现，项目中感觉不到模型的存在，我们的项目更依赖人员经验将业务需求转化为代码实现。我这个土豹子搬着小板...

展开

作者回复: 可以尝试做模型提取

1

林铭铭

2021-07-21

看完这一讲，突然想起来之前做金融类系统，需求文档开头会把涉及的概念和术语定义出来，我觉得这个应该算是最初版本的统一语言。

展开

Geek_ea8a4f
2021-07-08

怎么把实体合理分成多个聚合。判断实体属于一个聚合的标准是什么

作者回复: 根据上下文需要

Todd BD
2021-07-06

请教一下老师 Ubiquitous language 和 Domain Specific Language 有什么区别和联系？

作者回复: dsl不需要全体共识 只要靠近领域即可

1